BUNDESREPUBLIK DEUTSCHLAND

Prioritätsbescheinigung über die Einreichung einer Patentanmeldung

Aktenzeichen: 102 44 747.0

Anmeldetag: 25. September 2002

Anmelder/Inhaber: Siemens Aktiengesellschaft, München/DE

Bezeichnung: Medizinische Systemarchitektur mit einer Vorrichtung zur Übertragung von Daten, Untersuchungs-Bildern und Nachrichten sowie Verfahren zum Austausch von Nachrichten

IPC: G 06 F 19/00

Die angehefteten Stücke sind eine richtige und genaue Wiedergabe der ursprünglichen Unterlagen dieser Patentanmeldung.

München, den 8. September 2003
Deutsches Patent- und Markenamt
Der Präsident
Im Auftrag

Stanschus

A 9161
03/00
EDV-L

Beschreibung

Medizinische Systemarchitektur mit einer Vorrichtung zur Übertragung von Daten, Untersuchungs-Bildern und Nachrichten sowie Verfahren zum Austausch von Nachrichten

Die Erfindung betrifft eine medizinische Systemarchitektur mit wenigstens einer Modalität zur Erfassung von Untersuchungs-Bildern, mit den jeweiligen Modalitäten zugeordneten Rechnerarbeitsplätzen zur Verarbeitung der Untersuchungs-Bilder, mit einer Vorrichtung zur Übertragung von Daten, den Untersuchungs-Bildern und Nachrichten zwischen Client-Applikationen und Server-Applikationen, mit einer Speichervorrichtung für die Daten und Untersuchungs-Bilder und mit weiteren Rechnerarbeitsplätzen zur Nachbearbeitung der Daten und Untersuchungs-Bilder sowie ein Verfahren zum Austausch von Nachrichten zwischen Knoten eines Netzwerkes.

Aus dem Buch "Bildgebende Systeme für die medizinische Diagnostik", herausgegeben von H. Morneburg, 3. Auflage, 1995, Seiten 684ff sind medizinische Systemarchitekturen, sogenannte PACS (Picture Archival and Communication Systeme), bekannt, bei denen zum Abruf von Patientendaten und durch Modalitäten erzeugter Bilder Bildbetrachtungs- und Bildbearbeitungsplätze, sogenannte Workstations, über ein Bildkommunikationsnetz miteinander verbunden sind. Mittels dieser Workstation lassen sich die Bilder durch Experten befunden.

Beim Zusammenwirken derartiger Systeme treten folgende technische Probleme auf:

a) DICOM Kompatibilitätsprobleme während Netzwerkkommunikation zwischen DICOM Knoten sowohl Vorwärts, Rückwärts als auch mit Produkten von anderen Herstellern werden generisch und konfigurierbar gelöst.
   Neue Systeme müssen berücksichtigen, wie sich Altsysteme oder andere Produkte verhalten. Deshalb werden kostbare „Patches" und viel Testaufwand benötigt.

b) Anonymisierung von Patientendaten und andere sicherheitsrelevante Anforderungen können konfigurierbar ohne Änderungen in vorhandenen DICOM Produkten gelöst werden.
Die Anonymisierung muss heute in den Produkten hardkodiert eingebaut werden.

c) „DICOM Messages" von und zu eingekauften Simulatoren und Testwerkzeuge können heute zur Laufzeit (on the fly) nicht kundenspezifisch ausgebildet (customized) werden z.B. ein HIS/RIS Simulator kann DICOM Felder mit Nullen füllen, aber nicht leer weiterschicken, Altsysteme jedoch senden unbekannte Felder als leer.
Dies ließe sich nur durch die Entwicklung erweiterter Simulator- oder Version für Testwerkzeuge beseitigen.

Die Erfindung geht von der Aufgabe aus, eine medizinische Systemarchitektur der eingangs genannten Art zu schaffen sowie ein obengenanntes Verfahren derart auszubilden, dass eine leichte Anpassung an die verschiedensten Gegebenheiten und Anforderungen bedingt beispielsweise durch unterschiedliche Komponenten ggf. auch unterschiedlicher Hersteller auf einfache Weise erreicht wird.

Die Aufgabe wird erfindungsgemäß dadurch gelöst, dass der Vorrichtung zur Übertragung von Daten, den Untersuchungs-Bildern und Nachrichten ein Proxy-Server zugeordnet ist, der eine Umsetzung der Nachrichten zwischen Client-Applikationen und Server-Applikationen gemäß festgelegter Transformationsregeln bewirkt. Dadurch wird erreicht, dass das Netzwerk die Nachrichten zwischen zwei Knoten erfasst, den Inhalt laut konfigurierbarer Regeln manipuliert und danach weiterschickt.

In vorteilhafter Weise kann der Proxy-Server beim Austausch von Daten, Untersuchungs-Bildern und Nachrichten nach dem DICOM-Standard eingesetzt werden.

Erfindungsgemäß kann dem Proxy-Server ein Speicher für die Transformationsregeln zugeordnet sein.

Es hat sich als vorteilhaft erwiesen, wenn der Proxy-Server eine separate Softwareapplikation ist.

Der Proxy-Server kann erfindungsgemäß auf demselben Knoten oder auf einem Netzwerkknoten laufen.

Die Aufgabe wird für ein Verfahren erfindungsgemäß dadurch gelöst, dass der Inhalt der Nachrichten bei ihrer Übertragung mittels einer Umsetzungsroutine nach Transformationsregeln manipuliert werden, wobei der Austausch der Nachrichten zwischen Client-Applikationen und Server-Applikationen erfolgt.

In vorteilhafter Weise können die Applikationen DICOM-Applikationen sein.

Es hat sich als vorteilhaft erwiesen, wenn die Transformationsregeln konfigurierbar sind, so dass eine leichte Anpassung an die verschiedensten Gegebenheiten und Anforderungen erreicht werden kann.

Erfindungsgemäß kann die Umsetzung von Nachrichten durch einen Proxy-Server durchgeführt werden, der auf gespeicherte Transformationsregeln zugreift, wobei der Empfang, die Manipulation und das Weiterschicken der Nachrichten für die DICOM Knoten transparent sind.

Die Erfindung ist nachfolgend anhand von in der Zeichnung dargestellten Ausführungsbeispielen näher erläutert. Es zeigen:

Figur 1 ein Beispiel einer Systemarchitektur eines Krankenhausnetzes,

Figur 2 schematisch eine bekannte Kommunikation zwischen einer DICOM-Client-Applikation und einer DICOM-Server-Applikation und

Figur 3 die erfindungsgemäße Kommunikation zwischen einer DICOM-Client-Applikation und einer DICOM-Server-Applikation.

In der Figur 1 ist beispielhaft die Systemarchitektur eines Krankenhausnetzes dargestellt. Zur Erfassung medizinischer Bilder dienen die Modalitäten 1 bis 4, die als bilderzeugende Systeme beispielsweise eine CT-Einheit 1 für Computertomographie, eine MR-Einheit 2 für Magnetische Resonanz, eine DSA-Einheit 3 für digitale Subtraktionsangiographie und eine Röntgeneinheit 4 für die digitale Radiographie aufweisen kann. An diese Modalitäten 1 bis 4 sind Bedienerkonsolen 5 bis 8 der Modalitäten oder Workstations angeschlossen, mit denen die erfassten medizinischen Bilder verarbeitet und lokal abgespeichert werden können. Auch lassen sich zu den Bildern gehörende Patientendaten eingeben.

Die Bedienerkonsolen 5 bis 8 sind mit einem Kommunikationsnetz 9 als LAN/WAN Backbone zur Verteilung der erzeugten Bilder und Kommunikation verbunden. So können beispielsweise die in den Modalitäten 1 bis 4 erzeugten Bilder und die in den Bedienerkonsolen 5 bis 8 weiter verarbeiteten Bilder in zentralen Bildspeicher- und Bildarchivierungssystemen 10 abgespeichert oder an andere Workstations weitergeleitet werden.

An dem Kommunikationsnetz 9 sind weitere Viewing-Workstations 11 als Befundungskonsolen angeschlossen, die lokale Bildspeicher aufweisen. Eine derartige Viewing-Workstation 11 ist beispielsweise ein sehr schneller Kleincomputer auf der Basis eines oder mehrerer schneller Prozessoren. In den Viewing-Workstations 11 können die erfassten und im Bildarchivierungssystem 10 abgelegten Bilder nachträglich zur Befundung abgerufen und in dem lokalen Bildspeicher abgelegt werden,

von dem sie unmittelbar der an der Viewing-Workstation 11 arbeitenden Befundungsperson zur Verfügung stehen können.

Weiterhin sind an dem Kommunikationsnetz 9 Server 12, beispielsweise Patientendaten-Server (PDS), Fileserver, Programm-Server und/oder EPR-Server angeschlossen.

Der Bild- und Datenaustausch über das Kommunikationsnetz 9 erfolgt dabei nach dem DICOM-Standard, einem Industriestandard zur Übertragung von Bildern und weiteren medizinischen Informationen zwischen Computern, damit eine digitale Kommunikation zwischen Diagnose- und Therapiegeräten unterschiedlicher Hersteller möglich ist. An dem Kommunikationsnetz 9 kann ein Netzwerk-Interface 13 angeschlossen sein, über das das interne Kommunikationsnetz 9 mit einem globalen Datennetz, beispielsweise dem World Wide Web verbunden ist, so dass die standardisierten Daten mit unterschiedlichen Netzwerken weltweit ausgetauscht werden können.

An dem Kommunikationsnetz 9 kann weiterhin ein RIS- und/oder KIS-Server 14 angeschlossen sein, mit dem die Bedienerkonsolen 5 bis 8 mittels des Kommunikationsnetzes 9 über TCP/IP-Protokolle kommunizieren.

In Figur 2 ist eine normale Kommunikation zwischen einer Applikation 15 auf einem DICOM-Client, beispielsweise auf einer der Modalitäten 1 bis 4, und einer Applikation 16 auf einem DICOM-Server, beispielsweise auf dem Server 14, schematisch dargestellt. In einer ersten Verbindung werden mehrere Nachrichten 17 ausgetauscht, die direkt vom DICOM-Client zum DICOM-Server und zurück gehen.

Figur 3 zeigt nun eine erfindungsgemäße Kommunikation zwischen einer Client-Applikation 15 und einer Server-Applikation 16. Die Nachrichten 17 vom DICOM-Client zum DICOM-Server und zurück werden erst einem sogenannten Proxy-Server 18 zu-

geführt, der sie mit in einem Speicher 19 abgelegten Transformationsregeln umsetzt.

Der Proxy-Server 18 ist eine Komponente, die den Datenverkehr im Internet für ein lokales Netzwerk (LAN) verwaltet.

Dieser Proxy-Server 18 kann eine separate Softwareapplikation sein. Sie kann auf demselben Knoten oder auf einem Netzwerkknoten laufen. Sie ist regelbasiert und sehr dynamisch konfigurierbar. Sie ist semantisch frei.

Durch den Proxy-Server 18 werden im Kommunikationsnetz 9 die Daten zwischen den DICOM Knoten erfasst, deren Inhalt nach konfigurierbaren Regeln manipuliert und danach an den Empfänger weiterschickt. Der Empfang, die Manipulation und das Weiterschicken ist für die DICOM Knoten analog zu einer HTTP Proxy total transparent.

Die Manipulation erhält eine hohe Ausdruckskraft durch den mächtigen „Regular Expression Pattern Matching" Algorithmus, der von dem Matematiker S. Kleene stammt und beispielsweise in dem Buch "Mastering Regular Expressions . Powerful techniques for Perl and other tools" von Jeffrey E.F. Friedl beschrieben ist. Er wird benutzt, um Muster von Strings eindeutig und mit einem starken algebraischen Grund zu beschreiben. Die "Regular Expression"-Muster sind ausfaktorisiert in Konfigurationsdateien. Es ist keine Übersetzung des Quellcodes notwendig, um den Proxy umprogrammieren zu können.

Die medizinische Systemarchitektur gemäß der Erfindung zeichnet sich durch folgende Ausprägungen aus:

- Transparenter Proxy zwischen einer alten Generation von Dicom-basierten Produkten und einem neuen Produkt.

- Transparenter Proxy zwischen Dicom-Produkten verschiedener Hersteller oder Interpretationen.

- Sicherheits-Firewall zu anderen Netzwerken oder Dicom-Knoten.

- Erweiterung von Dicom-Simulatoren oder Interoperabilitäts-Testwerkzeuge.

Anhang 1

In der Beschreibung verwendete Abkürzungen:

DICOM	Digital Imaging and Communications in Medicine DICOM-Standard ist ein Industriestandard zur Übertragung von Bildern und weiteren medizinischen Informationen zwischen Computern zur Ermöglichung der digitalen Kommunikation zwischen Diagnose- und Theraphiegeräten unterschiedlicher Hersteller.
EPR	Electronic-Patient-Record (Elektronische Patienten Akte)
HIS (KIS)	Hospital Information System (Krankenhaus Information System): System für allgemeines Krankenhaus Management, mit den Hauptmerkmalen Patienten Management, Buchhaltung und Rechnungswesen, Personal Management usw.
HTTP	HyperText Transfer Protocol definiert den Zugriff von Clients, z. B. Webbrowsern, auf serverseitig gespeicherte Informationen im World Wide Web. HTTP definiert, wie Nachrichten formatiert und übertragen werden und welche Aktionen Webserver und Webbrowser als Antwort auf verschiedene Befehle durchführen sollen.
LAN	Local Area Network Ein lokales Netzwerk, das aus einer Gruppe von Computern und anderen Geräten besteht, die über einen relativ begrenzten Bereich verteilt und durch Kommunikationsleitungen verbunden sind, die jedem Gerät die Interaktion mit jedem anderen Gerät im Netzwerk ermöglichen.

PACS — Picture Archival and Communication System: computergestützte Bild-Informationssysteme zur Optimierung der Patientenversorgung, des Arbeitsablaufes in der radiologischen Abteilung, der Bildverteilung im Krankenhaus, der Bildversorgung für Forschung und Lehre und der Bildarchivierung.

RIS — Radiologie Informationssystem (Radiology Information System): Information System zum Daten-Management innerhalb der Radiologie Abteilung, das beispielsweise den Patienten Zugang, die Kreation von Worklisten, das Berichtswesen, Report Management, die Buchhaltung und das Rechnungswesen usw. unterstützt.

TCP/IP — Transmission Control Protocol/Internet Protocol (Übertragungssteuerungsprotokoll/Internetprotokoll). Das Protokoll für die Kommunikation zwischen Computern ist in das Betriebssystem UNIX integriert und ein De-facto-Standard für die Datenübertragung über Netzwerke, einschließlich dem Internet.

WAN — Weitbereichsnetz, (wide area network) Ein Kommunikationsnetzwerk zur Verbindung geografisch weit getrennter Regionen. Ein Weitbereichsnetz kann aus mehreren lokalen Netzwerken besteht. Ein Beispiel für ein Weitbereichsnetzwerk ist das Internet.

Anhang 2

```
// ------------------------------------------------------------

//   Schematic example code for a tool converting from an
//   ascii based file using the minimal language
//   (set-content-from-string,set-content-from-file,
//   open-item, close-item) with some cosmetic extensions

// ------------------------------------------------------------

#include <stdio.h>
#include <string.h>

// include some-dicom-toolkit-header-files
// for dicom stream build and dicom constrains checking

// ------------------------------------------------------------
// Globals
#define BIGSTRSIZE 1000
#define BIGPNAMESIZE 500
#define BUFFERSIZE 1024*1024*10
#define VRSIZE 3

#define SET_FROM_STR 's'
#define SET_FROM_FUN 'f'
#define OPEN_ITEM 'O'
#define CLOSE_ITEM 'C'
#define TOOLKIT_DUMP 'D'
#define DEBUGTOGGLE '!'
#define COMMENT_BEGIN '#'
#define QUIT 'q'

// Toolkit data structures should be added here

char        appName[10] = "foobar";
char*       fileName;
```

```
int lineNo = 0; // line number in input command file

typedef struct line_t
{
  int cmd;
  unsigned long tag;
  char value[BIGSTRSIZE];
  unsigned long slot1;       // DICOM group value
  char slot2[BIGPNAMESIZE]; // DICOM private tag: OwnerCode
  unsigned long slot3;       // DICOM element value or DICOM
          private tag: ElementByte
  char slot4[VRSIZE];        // DICOM private tag: value rep-
          resentation

} line_t;

line_t line;

char callbackdata = 0x1;


// Stack for message id
#define STACKMAXVAL 10000
int msgId_sp=0;
int msgId_val[STACKMAXVAL];

// ------------------------------------------------------------
// Functions

void pushmsgId(int v){
  if (msgId_sp < STACKMAXVAL)
    msgId_val[msgId_sp++] = v;
  else
    fprintf(stderr, "Schematic-example-code at cmdline %d:
          Stack overflow\n",lineNo);
}
```

```
int popmsgId(void){
  if (msgId_sp>0)
    return msgId_val[--msgId_sp];
  else {
    fprintf(stderr, "Schematic-example-code at cmdline %d:
          Stack underflow\n",lineNo);
    return 0;
  }
}

void ErrExit(char * errMsg, int errNum)
{
  if (errNum == NORMAL_COMPLETION) {
    fprintf(stderr, "Schematic-example-code at cmdline %d:
          %s\n", lineNo, errMsg);
  } else {
    fprintf(stderr, "Schematic-example-code at cmdline %d: %s
          with TOOLKIT error %s\n", lineNo, errMsg, Er-
          ror_Message( (STATUS) errNum));
   }
  status = Release_Application( &appID);
  exit(-1);
}

static STATUS simpleCallBack (
                    int             msgID,
                    unsigned long   tag,
                    int             firstCall,
                    void*           userInfo,
                    int*            dataLen,
                    void**          dataBuffer,
                    int*            isLast)
{
    static char    buffer[BUFFERSIZE];
    size_t          byte_pos;
    FILE *stream;
```

```
char errMsg[100];
int ch;

*isLast = 1;


if( (stream  = fopen( line.value, "rb" )) == NULL ) {
  sprintf( errMsg,"Failed to open file: '%s'\n",
      line.value );
  ErrExit(errMsg,NORMAL_COMPLETION);
}

byte_pos = 0;
ch = fgetc( stream );

while ( feof( stream ) == 0 ) {
  buffer[byte_pos] = ch;
  ch = fgetc( stream );
  byte_pos++;
}
// byte_pos points one ahead of last filled pos in buffer
      and equals number of bytes

if( fclose( stream ) ) {
  sprintf( errMsg,"Failed to close file: '%s'\n",
      line.value );
  ErrExit(errMsg,NORMAL_COMPLETION);
}

// if odd number of bytes report error
if ((byte_pos % 2) == 1) {
   ErrExit("Odd number of bytes
      read.\n",NORMAL_COMPLETION);
}
```

```
    *dataBuffer = buffer;
    *dataLen = (int)byte_pos;

    return NORMAL_COMPLETION;
}

VR str2vr(char * str)
{

  if ((str[0] == 'A') && (str[1] == 'E')){return AE;}
  if ((str[0] == 'A') && (str[1] == 'S')){return AS;}
  if ((str[0] == 'C') && (str[1] == 'S')){return CS;}
  if ((str[0] == 'D') && (str[1] == 'A')){return DA;}
  if ((str[0] == 'D') && (str[1] == 'S')){return DS;}
  if ((str[0] == 'D') && (str[1] == 'T')){return DT;}
  if ((str[0] == 'I') && (str[1] == 'S')){return IS;}
  if ((str[0] == 'L') && (str[1] == 'O')){return LO;}
  if ((str[0] == 'L') && (str[1] == 'T')){return LT;}
  if ((str[0] == 'P') && (str[1] == 'N')){return PN;}
  if ((str[0] == 'S') && (str[1] == 'H')){return SH;}
  if ((str[0] == 'S') && (str[1] == 'T')){return ST;}
  if ((str[0] == 'T') && (str[1] == 'M')){return TM;}
  if ((str[0] == 'U') && (str[1] == 'T')){return UT;}
  if ((str[0] == 'U') && (str[1] == 'I')){return UI;}
  if ((str[0] == 'S') && (str[1] == 'S')){return SS;}
  if ((str[0] == 'U') && (str[1] == 'S')){return US;}
  if ((str[0] == 'A') && (str[1] == 'T')){return AT;}
  if ((str[0] == 'S') && (str[1] == 'L')){return SL;}
  if ((str[0] == 'U') && (str[1] == 'L')){return UL;}
  if ((str[0] == 'F') && (str[1] == 'L')){return FL;}
  if ((str[0] == 'F') && (str[1] == 'D')){return FD;}
  if ((str[0] == 'O') && (str[1] == 'B')){return OB;}
  if ((str[0] == 'O') && (str[1] == 'W')){return OW;}
  if ((str[0] == 'O') && (str[1] == 'L')){return OL;}
  if ((str[0] == 'S') && (str[1] == 'Q')){return SQ;}
```

```
 return UNKNOWN_VR;
}


void trim_rest_of_line(void)
{
  int c;

  c = getchar();
  while (c != '\n') {
     if (c == EOF) {
    return;
     }
    c = getchar();
  }
}


void trim(void)
{
  int c;

  c = getchar();
  while ((c == ' ') ||
     (c == '\t')) {
    c = getchar();
  }
  ungetc(c,stdin);
}


int parseLine(line_t *line)
{
   enum{CMD, TAG, VALUE} state = CMD;
   int c,i;
   char tagAsStr[BIGSTRSIZE];
```

```
// clear return data

line->cmd = 0;
line->tag = 0;

for(i=0; i < BIGSTRSIZE; i++) {
   line->value[i] = '\0';
}

line->slot1 = 0;
line->slot3 = 0;

for(i=0; i < BIGPNAMESIZE; i++) {
   line->slot2[i] = '\0';
}

for(i=0; i < VRSIZE; i++) {
   line->slot4[i] = '\0';
}

for(i=0; i<BIGSTRSIZE; i++) {
   tagAsStr[i] = '\0';
}

trim();


// get the command

c = getchar();
line->cmd = c;

if ((c == EOF) || (c == QUIT)) {
  return EOF;
}
```

```
if (c == COMMENT_BEGIN) {
  trim_rest_of_line();
  return COMMENT_BEGIN;
}

if (c == TOOLKIT_DUMP) {
  trim_rest_of_line();
  return TOOLKIT_DUMP;
}

if (c == DEBUGTOGGLE) {
  trim_rest_of_line();
  return DEBUGTOGGLE;
}

if (!((c == SET_FROM_STR) ||
  (c == SET_FROM_FUN) ||
  (c == TOOLKIT_DUMP) ||
  (c == OPEN_ITEM) ||
  (c == CLOSE_ITEM))) {
  ErrExit("Failed to parse line, unexpected
       cmd\n",NORMAL_COMPLETION);
}

trim();


// get the first slot of the tag.
// Tag format:
//       (0008,0090)
//       (0009,SIEMENS CM VA0  CMS,11)

if ((c = getchar()) != '(') {
  ErrExit("Failed to parse line, tag must begin with a
       parantesis\n",NORMAL_COMPLETION);
}
```

```
for(i=0; i<4; i++) {
  c = getchar();
  if (((c >= '0') && (c <= '9')) ||
  ((c >= 'A') && (c <= 'F')))
   {
 tagAsStr[i] = c;
   } else
   {
  ErrExit("Failed to parse line, invalid
       tag\n",NORMAL_COMPLETION);
   }
}

if (sscanf(tagAsStr,"%X",&line->slot1) != 1) {
  ErrExit("sscanf failed\n",NORMAL_COMPLETION);
}

for(i=0; i<BIGSTRSIZE; i++) {
  tagAsStr[i] = '\0';
}


if ((c = getchar()) != ',') {
  ErrExit("Failed to parse line, missing comma after group
       element in tag\n",NORMAL_COMPLETION);
}


// get the second slot
// if a comma follows its private if endparantesis its a
       normal tag, else error

c = getchar();
i = 0;
while (!((c == ',') ||
     (c == ')'))) {
```

```
        if ((c == '\n') ||
        (c == EOF)) {
          ErrExit("Failed to parse line, unexpected end in
             tag\n",NORMAL_COMPLETION);
        }

        tagAsStr[i++] = c;
        c = getchar();
      }

      if (c == ')') { // nonprivate tag

        for(i=0; i<4; i++) {
          if (!(((tagAsStr[i] >= '0') && (tagAsStr[i] <= '9'))
             ||
            ((tagAsStr[i] >= 'A') && (tagAsStr[i] <= 'F')))) {

        ErrExit("Failed to parse line, invalid hex value in sec-
             ond part of nonprivate tag\n",NORMAL_COMPLETION);
         }
        }

        if (sscanf(tagAsStr,"%X",&line->slot3) != 1) {
          ErrExit("sscanf failed, expected hex as second and
             last part of tag\n",NORMAL_COMPLETION);
        }

      } else { // private tag

        strcpy(line->slot2,tagAsStr);

        if (c != ',') {
          ErrExit("Failed to parse line, comma after slot2 miss-
             ing in private tag\n",NORMAL_COMPLETION);
        }
```

```
for(i=0; i<BIGSTRSIZE; i++) {
  tagAsStr[i] = '\0';
}

// slot3
c = getchar();
i = 0;
while (c != ',') {

  if ((c == '\n') ||
  (c == EOF)) {
ErrExit("Failed to parse line, unexpected end in slot3
     in private tag\n",NORMAL_COMPLETION);
  }

  tagAsStr[i++] = c;
  c = getchar();
}

// slot4
line->slot4[0] = getchar();
line->slot4[1] = getchar();

if ((c = getchar()) != ')') {
  ErrExit("Failed to parse line, missing ending parante-
     sis in private tag\n",NORMAL_COMPLETION);
}

if (sscanf(tagAsStr,"%X",&line->slot3) != 1) {
  ErrExit("sscanf failed\n",NORMAL_COMPLETION);
}
}


if ((line->cmd == SET_FROM_STR) || (line->cmd ==
     SET_FROM_FUN))
{
```

```
      c = getchar();
      if (!((c == ' ') || (c == '\t')))
      {
        ErrExit("Failed to parse line, delimiter before value
           not space or tab\n",NORMAL_COMPLETION);
      }
    }
    else
    {
      trim();
    }

    // get the value
    c = getchar();
    i = 0;
    while (!((c == EOF) ||
         (c == '\n'))) {
       line->value[i++] = c;
       c = getchar();
    }

    return c; // EOF or '\n'
}


int
main(int argc, char * argv[])
{
  int outer_msgId;
  bool debugging = FALSE;

  if (argc != 2)
    {
      fprintf(stderr, "\nSchematic-example-code Usage: Sche-
          matic-example-code filename < commands\n\n");
      fprintf(stderr,  "List of commands:\n\n");
```

```
        fprintf(stderr, "\t's' for SET_FROM_STR \n");
        fprintf(stderr, "\t'f' for SET_FROM_FUN \n");
        fprintf(stderr, "\t'O' for OPEN_ITEM \n");
        fprintf(stderr, "\t'C' for CLOSE_ITEM \n");
        fprintf(stderr, "\t'D' for TOOLKIT_DUMP \n");
        fprintf(stderr, "\t'!' for DEBUGTOGGLE \n");
        fprintf(stderr, "\t'#' for COMMENT_BEGIN \n");
        fprintf(stderr, "\t'q' for QUIT \n");

        return -1;
      }

    fileName = argv[1];

    status = Library_Initialization(NULL, NULL, NULL);
    if (status != NORMAL_COMPLETION)
      ErrExit("Library_Initialization", status);

    status = Register_Application(&appID, appName);
    if (status != NORMAL_COMPLETION)
      ErrExit("Register_Application", status);

    //status  =  Create_File (&msgId, fileName, "DICOMDIR",
            C_STORE_RQ);
    status  =  Create_Empty_File (&msgId, fileName);
    if (status != NORMAL_COMPLETION)
      ErrExit("Create_File", status);

    top_msgId = msgId;

    while (parseLine(&line) != EOF) {

       lineNo++;

       if ((line.cmd == QUIT) ||
       (line.cmd == COMMENT_BEGIN) ||
       (line.cmd == DEBUGTOGGLE) ||
```

```
(line.cmd == EOF)) {

  if (debugging) {
fprintf(stdout, "Schematic-example-code at cmdline %d:
    parsed cmd =  \"%c\" OK!\n", lineNo, line.cmd);
  }

} else {

  if (line.slot2[0] == '\0') {

  if (debugging) {
fprintf(stdout, "Schematic-example-code at cmdline %d:
    parsed cmd =  \"%c\" tag = (%4X,%4X) value \"%s\"
    OK!\n",
   lineNo,line.cmd,line.slot1,line.slot3,line.value);
  }

line.tag = (line.slot1 << 16) + line.slot3;

  } else {

if (debugging) {
  fprintf(stdout, "Schematic-example-code at cmdline %d:
    parsed cmd =  \"%c\" tag = (%4X,%s,%4X,%s) value
    \"%s\" OK!\n",

    lineNo,line.cmd,line.slot1,line.slot2,line.slot3,li
    ne.slot4,line.value,lineNo);
}
  }
}


switch (line.cmd) {

case SET_FROM_STR:
```

```
        if (line.slot2[0] == '\0')  // standard tag
        {
          if (strcmp(line.value, "") == 0)
          {
        status  = Set_Next_Value_To_NULL(msgId, line.tag);
            if (status == INCOMPATIBLE_VR)
            {
              status  = Set_Next_Value_From_String(msgId,
         line.tag, "");
            }
          }
          else
          {
        status  = Set_Next_Value_From_String(msgId, line.tag,
           line.value);
          }
      if ((status != NORMAL_COMPLETION) && (status !=
           INVALID_CHARS_IN_VALUE) && (status !=
           INVALID_VALUE_FOR_VR))
        ErrExit("Set_Next_Value_From_String", status);

        }
        else                              // private tag
        {

      unsigned long aValLength;
      status = Get_pValue_Length(msgId,
                     line.slot2,
                     (unsigned short) line.slot1,
                     (unsigned char) line.slot3,
                     1,
                     &aValLength);
         if ((status == NULL_VALUE) || (status ==
          EMPTY_VALUE))
         {
           status = NORMAL_COMPLETION;
```

```
    }

  if (!((status == NORMAL_COMPLETION) ||
        (status == INVALID_PRIVATE_CODE) ||
        (status == INVALID_TAG))) {
    ErrExit("Get_pValue_Length", status);
  }

  // CASE no owner element; no data element:
  if (status == INVALID_PRIVATE_CODE) {
    status = Add_Private_Block(msgId,line.slot2,(unsigned
       char)line.slot1);
    if (status == NORMAL_COMPLETION) {

      status = Add_Private_Attribute(msgId,line.slot2,
                      (unsigned short)line.slot1,
                      (unsigned char)line.slot3,
                      str2vr(line.slot4));
      if (status != NORMAL_COMPLETION) {
        ErrExit("Add_Private_Attribute failed", status);
      }
    } else {
      ErrExit("Add_Private_Block failed", status);
    }
  } else if (status == INVALID_TAG) {

    // CASE owner element exist; no data element: do 2.

    status = Add_Private_Attribute(msgId,line.slot2,
                        (unsigned short)line.slot1,
                        (unsigned char)line.slot3,
                        str2vr(line.slot4));
    if (status != NORMAL_COMPLETION) {
      ErrExit("Add_Private_Attribute\n", status);
    }
  } else if (status == NORMAL_COMPLETION) {
```

```
// CASE owner element exist; data element exist: do 3.

status = Set_pValue_Representation(msgId,line.slot2,
                        (unsigned short)line.slot1,
                        (unsigned char)line.slot3,
                        str2vr(line.slot4));
if (!((status == NORMAL_COMPLETION) ||
  (status == VR_ALREADY_VALID))) {
  ErrExit("Set_pValue_Representation", status);
}
} else {
  ErrExit("Other error while doing private attribute",
     status);
}
// We have possibly built a private structure, now set
     the private value
    if (strcmp(line.value, "") == 0)
    {
status  = Set_Next_pValue_To_NULL(msgId,
                        line.slot2,
                        (unsigned short)line.slot1,
                        (unsigned char)line.slot3);
    if (status == INCOMPATIBLE_VR)
    {
  status  = Set_Next_pValue_From_String(msgId,
                        line.slot2,
                        (unsigned short)line.slot1,
                        (unsigned char)line.slot3,
                        "");
    }
  }
  else
  {
status  = Set_Next_pValue_From_String(msgId,
                        line.slot2,
                        (unsigned short)line.slot1,
                        (unsigned char)line.slot3,
```

```
                line.value);
        }
      if ((status != NORMAL_COMPLETION) && (status !=
           INVALID_CHARS_IN_VALUE) && (status !=
           INVALID_VALUE_FOR_VR))
        {
        ErrExit("Set_Next_pValue_From_String", status);
        }
      }
      break;


    case SET_FROM_FUN:
      if (line.slot2[0] == '\0') { // standard tag

    status  = Set_Value_From_Function(msgId, line.tag, NULL,
        simpleCallBack);
    if (status != NORMAL_COMPLETION)
      ErrExit("Set_Value_From_Function", status);

      } else {

    unsigned long aValLength;
    status = Get_pValue_Length(msgId,
                    line.slot2,
                    (unsigned short) line.slot1,
                    (unsigned char) line.slot3,
                    1,
                    &aValLength);

    if (!((status == NORMAL_COMPLETION) ||
          (status == INVALID_PRIVATE_CODE) ||
          (status == INVALID_TAG))) {
      ErrExit("Get_pValue_Length", status);
    }

    // CASE no owner element; no data element:
```

```
if (status == INVALID_PRIVATE_CODE) {
  status = Add_Private_Block(msgId,line.slot2,(unsigned
     char)line.slot1);
  if (status == NORMAL_COMPLETION) {

    status = Add_Private_Attribute(msgId,line.slot2,
                    (unsigned short)line.slot1,
                    (unsigned char)line.slot3,
                    str2vr(line.slot4));
    if (status != NORMAL_COMPLETION) {
      ErrExit("Add_Private_Attribute failed", status);
    }
  } else {
    ErrExit("Add_Private_Block failed", status);
  }
} else if (status == INVALID_TAG) {

  // CASE owner element exist; no data element: do 2.

  status = Add_Private_Attribute(msgId,line.slot2,
                    (unsigned short)line.slot1,
                    (unsigned char)line.slot3,
                    str2vr(line.slot4));
  if (status != NORMAL_COMPLETION) {
    ErrExit("Add_Private_Attribute\n", status);
  }
} else if (status == NORMAL_COMPLETION) {

  // CASE owner element exist; data element exist: do 3.

  status = Set_pValue_Representation(msgId,line.slot2,
                    (unsigned short)line.slot1,
                    (unsigned char)line.slot3,
                    str2vr(line.slot4));
  if (!((status == NORMAL_COMPLETION) ||
    (status == VR_ALREADY_VALID))) {
    ErrExit("Set_pValue_Representation", status);
```

```
        }
      } else {
        ErrExit("Other error while doing private attribute",
           status);
      }
      // We have possibly built a private structure, now set
           the private value
      if ((status  = Set_pValue_From_Function(msgId,
                                 line.slot2,
                                 (unsigned short)line.slot1,
                                 (unsigned char)line.slot3,
                                 NULL,
                                 simpleCallBack)) !=
           NORMAL_COMPLETION) {
        ErrExit("Set_pValue_From_Function", status);
      }
        }
        break;

      case CLOSE_ITEM:
        outer_msgId = popmsgId();
        Set_Next_Value_From_Int(outer_msgId, line.tag, msgId);
        msgId = outer_msgId;
        if (status != NORMAL_COMPLETION)
      ErrExit("Set_Value_from_int", status);
        break;

      case TOOLKIT_DUMP:
     List_File(msgId, NULL);
     if (status != NORMAL_COMPLETION)
      ErrExit("List_File", status);
        break;

      case OPEN_ITEM:
        pushmsgId(msgId);
        status = Open_Item(&msgId, line.value);
        if (status != NORMAL_COMPLETION)
```

```
        ErrExit("Open_Item", status);
          break;

        case COMMENT_BEGIN:
          // do nothing for comments
          break;

        case DEBUGTOGGLE:
          if (debugging) {
        debugging = FALSE;
          } else {
        debugging = TRUE;
          }
          break;

        default:
          fprintf(stderr, "Schematic-example-code at cmdline %d:
             \"%c\": ",lineNo,line.cmd);
          ErrExit("Unexpected value",NORMAL_COMPLETION);
          break;
        }
      }

      if (debugging) {
        fprintf(stdout, "Schematic-example-code: Writing dicom
             output to: %s\n", fileName);
      }


      long aStatus = UNDEFINED;
      TransferSyntax aSyntax;
      const int aBufferSize = 128;
      char aBuffer[aBufferSize];

      status = Get_Value_To_String(msgId, 0x00020010, aBuffer-
             Size, aBuffer);
      if (status == NORMAL_COMPLETION)
```

```
    {
      aSyntax.setFromUidString(aBuffer);
    }
    else
    {
      status = File_To_Message(msgId);
      if (status != NORMAL_COMPLETION)
      {
        ErrExit("File_To_Message", status);
      }
      aSyntax.setFromString("STREAM_IMPLICIT_LITTLE_ENDIAN");
    }

    aStatus = encodeDicomList(msgId, fileName, aSyntax);
    if (aStatus != SUCCESS)
    {
      fprintf(stderr, "Schematic-example-code at cmdline %d:
            %s\n", lineNo, "encodeDicomList() failed");
    }

    return 0;
  } // main
```

Patentansprüche

1. Medizinische Systemarchitektur mit wenigstens einer Modalität (1 bis 4) zur Erfassung von Untersuchungs-Bildern, mit den jeweiligen Modalitäten (1 bis 4) zugeordneten Rechnerarbeitsplätzen (5 bis 8) zur Verarbeitung der Untersuchungs-Bilder, mit einer Vorrichtung (9) zur Übertragung von Daten, den Untersuchungs-Bildern und Nachrichten zwischen Client-Applikationen (15) und Server-Applikationen (16), mit einer Speichervorrichtung (10) für die Daten und Untersuchungs-Bilder und mit weiteren Rechnerarbeitsplätzen (11) zur Nachbearbeitung der Daten und Untersuchungs-Bilder, wobei der Vorrichtung (9) ein Proxy-Server (18) zugeordnet ist, der eine Umsetzung der Nachrichten zwischen Client-Applikationen (15) und Server-Applikationen (16) gemäß festgelegter Transformationsregeln bewirkt.

2. Medizinische Systemarchitektur nach Anspruch 1, dadurch gekennzeichnet, dass der Austausch von Daten, Untersuchungs-Bildern und Nachrichten nach dem DICOM-Standard erfolgt.

3. Medizinische Systemarchitektur nach Anspruch 1 oder 2, dadurch gekennzeichnet, dass dem Proxy-Server (18) ein Speicher (19) für die Transformationsregeln zugeordnet ist.

4. Medizinische Systemarchitektur nach einem der Ansprüche 1 bis 3, dadurch gekennzeichnet, dass der Proxy-Server (18) eine separate Softwareapplikation ist.

5. Medizinische Systemarchitektur nach einem der Ansprüche 1 bis 4, dadurch gekennzeichnet, dass der Proxy-Server (18) auf demselben Knoten oder auf einem Netzwerkknoten läuft.

6. Verfahren zum Austausch von Nachrichten (17) zwischen Knoten eines Netzwerkes (9), d a d u r c h g e k e n n z e i c h n e t , dass der Inhalt der Nachrichten bei ihrer Übertragung mittels einer Umsetzungsroutine nach Transformationsregeln manipuliert werden.

7. Verfahren nach Anspruch 6, d a d u r c h g e k e n n z e i c h n e t , dass der Austausch von Nachrichten zwischen Client-Applikationen (15) und Server-Applikationen (16) erfolgt.

8. Verfahren nach Anspruch 6 oder 7, d a d u r c h g e k e n n z e i c h n e t , dass die Applikationen (15, 16) DICOM-Applikationen sind.

9. Verfahren nach einem der Ansprüche 6 bis 8, d a d u r c h g e k e n n z e i c h n e t , dass die Transformationsregeln konfigurierbar sind.

10. Verfahren nach einem der Ansprüche 6 bis 9, d a d u r c h g e k e n n z e i c h n e t , dass die Umsetzung von Nachrichten durch einen Proxy-Server (18) durchgeführt wird, der auf gespeicherte Transformationsregeln zugreift.

11. Verfahren nach einem der Ansprüche 6 bis 10, d a d u r c h g e k e n n z e i c h n e t , dass der Empfang, die Manipulation und das Weiterschicken der Nachrichten für die DICOM Knoten transparent sind.

Zusammenfassung

Medizinische Systemarchitektur mit einer Vorrichtung zur Übertragung von Daten, Untersuchungs-Bildern und Nachrichten sowie Verfahren zum Austausch von Nachrichten

Die Erfindung betrifft eine medizinische Systemarchitektur mit wenigstens einer Modalität (1 bis 4) zur Erfassung von Untersuchungs-Bildern, mit den jeweiligen Modalitäten (1 bis 4) zugeordneten Rechnerarbeitsplätzen (5 bis 8) zur Verarbeitung der Untersuchungs-Bilder, mit einer Vorrichtung (9) zur Übertragung von Daten, den Untersuchungs-Bildern und Nachrichten zwischen Client-Applikationen (15) und Server-Applikationen (16), mit einer Speichervorrichtung (10) für die Daten und Untersuchungs-Bilder und mit weiteren Rechnerarbeitsplätzen (11) zur Nachbearbeitung der Daten und Untersuchungs-Bilder, wobei der Vorrichtung (9) ein Proxy-Server (18) zugeordnet ist, der eine Umsetzung der Nachrichten zwischen Client-Applikationen (15) und Server-Applikationen (16) gemäß festgelegter Transformationsregeln bewirkt, sowie ein Verfahren zum Austausch von Nachrichten (17) zwischen Knoten eines Netzwerkes (9), wobei der Inhalt der Nachrichten bei ihrer Übertragung mittels einer Umsetzungsroutine nach Transformationsregeln manipuliert werden.

Figur 3

FIG 1

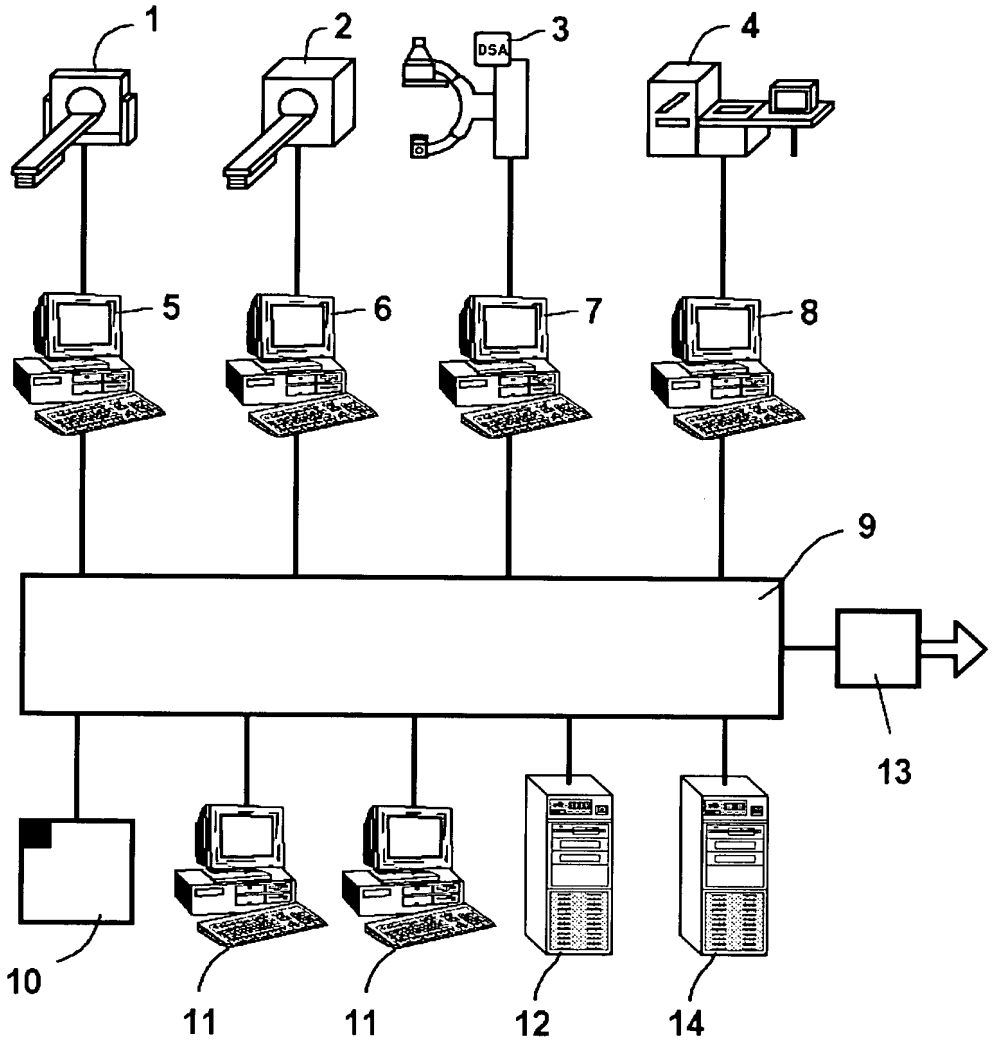

FIG 2

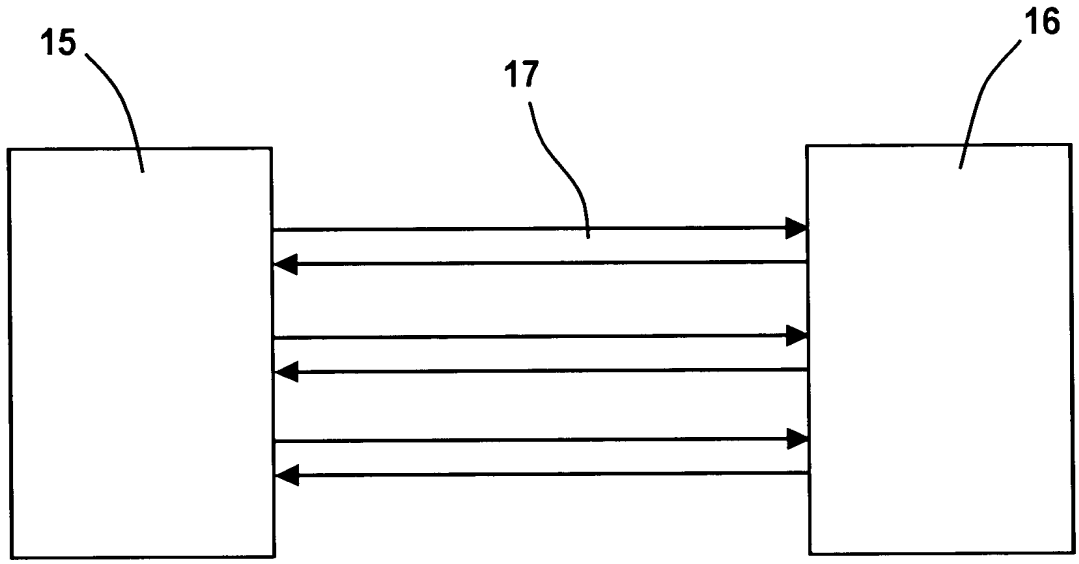

FIG 3